# Extending the $\mathcal{SHOIQ}(D)$ Tableaux with DL-safe Rules: First Results

Vladimir Kolovski and Bijan Parsia and Evren Sirin
University of Maryland
College Park, MD 20740
kolovski@cs.umd.edu
bparsia@isr.umd.edu
evren@cs.umd.edu

**Abstract**

On the Semantic Web, there has been increasing demand for a rules-like expressivity that goes beyond OWL-DL. Efforts of combining rules languages and description logics usually produce undecidable formalisms, unless constrained in a specific way. We look at one of the most expressive - but decidable - such formalisms proposed: DL-safe rules, and present a tableaux-based algorithm for query answering in OWL-DL augmented with a DL-safe rules component. In this paper we present our algorithm which is an extension of the $\mathcal{SHOIQ}$ tableaux algorithm and show the preliminary empirical results. Knowing that we pay a high price in performance for this expressivity, we also explore possible optimizations.

## 1 Introduction

Even though OWL-DL [3] has been a great success as a language for ontology representation on the Semantic Web, there is already demand for even greater expressivity in the form of logic programming (LP) rules. Community interest in this area is growing, as evidenced by the recent (November 2005) creation of the Rule Interchange Format Working Group (RIF [2]). One approach to provide the needed additional expressivity is to couple in some way Description Logic with LP rule systems (the hybrid approach).

However, combining expressive Description Logics with rules is non-trivial since it can easily lead to undecidability if the integration is not restricted in some manner. There have been a number of proposals that follow the hybrid approach while still retaining decidability [12, 4, 9]. The logic of DL-safe rules [12] is one of the most expressive of these, combining $\mathcal{SHIQ}(\mathcal{D})$ and positive datalog rules. DL-safe rules allow classes and properties from the Description Logic component to occur freely in the head or the body of the rule, with the only restriction being that they can be applied only to explicitly named individuals. The same authors who coined the term also proposed the only practical reasoning algorithm for this formalism [11]. The algorithm is based on reducing the description logic knowledge base to a positive disjunctive datalog program, appending the DL-safe rules to this program and using optimized deductive database techniques for query answering.

As an alternative to a reduction to disjunctive datalog, in this paper we present a direct tableaux algorithm for DL-safe rules, as an extension to the $\mathcal{SHOIQ}$ tableaux algorithm. Instead of converting the Description Logic knowledge base to a datalog program, we extend a highly optimized OWL-DL tableaux reasoner to handle these rules. We believe that we will benefit from layering our algorithm on top of an efficient tableaux reasoner and our initial empirical results support this claim. Also, this approach allows our algorithm to cover the full expressivity of OWL-DL ($\mathcal{SHOIN}(\mathcal{D})$) and $DL$-safe rules.

In the following section, we provide preliminaries and definitions of terms used later in this paper. Then, after presenting our algorithm, we discuss its performance in the preliminary tests, and possible optimizations and future work.

## 2 Background

In this section we provide a brief overview of the terms and notation used later in the paper. We assume that the reader is familiar with the syntax and semantics of $\mathcal{SHOIN}(D)$. Before introducing DL-safe rules, we need to describe the standard notation we will be using for rules. A *term* is either a constant (denoted by *a, b, c*) or a variable (denoted by *x, y, z*). An *atom* has the form $P(s_1, ..., s_n)$, where $P$ is a predicate symbol and $s_i$ are terms. A literal is an atom or a negated atom. A *rule* has the form

$$H \leftarrow B_1, \ldots, B_n$$

where $H$ and $B_i$ are atoms; $H$ is called the rule *head*, and the set of all $B_i$ is called the rule *body*. A *program* $P$ is a finite set of rules. As for the semantics, we treat rules as logical implications, thus they can also be represented as:

$$H \vee \neg B_1 \vee \neg B_2 \vee \ldots \neg B_n$$

**Definition 1** *(Semantics of a DL-safe Knowledge Base)*

*We start with a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base, $\mathcal{K}$. Let $V_C$, $V_{IP}, V_{DP}$ be countable and pair-wise disjoint sets of class, object property and datatype property names respectively. Let $V_P$ be a set of predicate symbols that can be used as atoms in the rules, such that $V_C \cup V_{IP} \cup V_{DP} \subseteq V_P$ . A DL-atom is an atom of the form $A(s)$ were $A$ belongs to $V_C$, or of the form $R(s,t)$ where $R$ belongs to $V_{IP} \cup V_{DP}$. A rule $r$ is called DL-safe if each variable in $r$ occurs in a non-DL-atom in the rule body. A program is DL-safe iff all of its rules are DL-safe.*

*Let $\mathcal{K}$ be a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base and $P$ a set of DL-safe rules. A* ***DL-safe Knowledge Base*** *is a pair $(\mathcal{K}, P)$. Since $\mathcal{SHOIN}(\mathbf{D})$ is a subset of first order logic, we can give the semantics of $(\mathcal{K}, P)$ by $\pi(\mathcal{K}) \cup P$ where $\pi(\mathcal{K})$ is a translation mapping $\mathcal{SHOIN}(\mathbf{D})$ to first order logic.*

## 3 Tableaux Algorithm

In this section we present a tableaux-based algorithm for deciding the satisfiability of DL-safe knowledge bases. Our algorithm is inspired by the decidability proof for query answering in combined $\mathcal{SHOIN}(D)$ and DL-safe rules [12]. The main difference is that we push the rule reasoning inside the tableaux algorithm in order to get some goal directed behavior. Our approach extends the $\mathcal{SHOIQ}$ algorithm [7] thus we are able to handle the full expressivity of OWL-DL and DL-safe rules. We assume that the reader is familiar with the basics of the $\mathcal{SHOIQ}$ algorithm (detailed description is available in [7]).

The main reasoning service that our algorithm provides is query answering. More specifically, answering for a given query $\alpha$ and a DL-safe knowledge base $KB = (\mathcal{K}, P)$ whether $KB \models \alpha$. This can be reduced to a consistency check of $KB \cup \{\neg\alpha\}$ , so $KB \models \alpha$ iff $KB \cup \{\neg\alpha\}$ is not consistent.

To check whether a DL-safe KB $(\mathcal{K}, P)$ is consistent, we need to perform a consistency check of its ABox with respect to the TBox, the RBox and the rules. Since it was shown in [7] that we can reduce reasoning w.r.t. general TBoxes and role hierarchies to reasoning w.r.t. role hierarchies only, from now on without loss of generality we will assume that we have internalized the TBox and we do reasoning w.r.t. to the RBox and the rules program.

Since ABoxes in general involve multiple individuals with arbitrary role relationships between them, the completion algorithm will work on a forest instead of a tree. Such a forest is a collection of trees whose root nodes correspond to the individuals present in the input ABox. The individuals in the input ABox are the explicitly asserted individuals, and are the *only* individuals to which the rules in $P$ are applicable. A formal description of a completion forest is given in the following definition.

**Definition 2** *($\mathcal{SHOIQ}$ + DL-safe Completion Forest) (extended version of [8]) A completion forest $\mathcal{F}$ for a $\mathcal{SHOIQ}$ Abox $\mathcal{A}$ w.r.t a role hierarchy $\mathcal{R}$ and a*

*DL-safe KB $(\mathcal{K}, P)$ is a collection of trees whose distinguished root nodes may be connected by edges in an arbitrary way. The completion forest can be described as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{L}, \mathcal{E}, P, \neq)$ where each individual $x \in \mathcal{V}$ can be labeled with a set of concept labels $\mathcal{L}(x)$ and each edge $(x, y)$ can be labeled with a set of role names $\mathcal{E}(x, y)$. $P$ is a program consisting of DL-safe rules $R_i$ ,each one of the form*

$$H_i \vee \neg B_{i1} \vee \neg B_{i2} \vee \ldots \neg B_{in}$$

*We also keep track of inequalities between nodes of the graph with a symmetric binary relation $\neq$ between the nodes of $\mathcal{G}$. Given a $\mathcal{SHOIQ}$ Abox $\mathcal{A}$, a role hierarchy $\mathcal{R}$ and a DL-safe program $P$, the algorithm initializes a completion forest $\mathcal{F}$ that contains a root node $x_0^i$ for each individual $a_i \in \mathcal{A}$, and an edge $(x_0^i, x_0^j)$ if $\mathcal{A}$ contains an assertion $(a_i, a_j) : R$ for a role $R$. The labels of these nodes and edges are initialized as follows:*

$$\mathcal{L}(x_0^i) = \{C | a_i : C \in \mathcal{A}\}$$
$$\mathcal{L}((x_0^i, x_0^j)) = \{R | (a_i, a_j) : R \in \mathcal{A}\}$$
$$x_0^i \neq x_0^j \textit{ iff } a_i \neq a_j \in \mathcal{A}$$

*For the purpose of our paper, we assume standard blocking and clash definitions [7].*

*The tableaux algorithm starts with the completion forest $\mathcal{F}$. It applies the completion rules and stops when a clash occurs. The ABox $\mathcal{A}$ is consistent w.r.t. to $\mathcal{R}$ and $P$ iff the completion rules can be applied in a way that they yield a complete and clash-free completion forest, and inconsistent otherwise.*

Our extension of the algorithm is in the form of a new completion rule, $R$-rule [1]. We reuse all of the original $\mathcal{SHOIQ}$ completion rules, and add the $R$-rule. The $R$-rule starts with generating all of the possible ground instantiations for a DL-safe rule (we will refer to them as *bindings* henceforth). We need to be careful with the bindings because only named individuals **not** introduced by the tableaux expansion rules are considered. For every such binding, we check whether at least one of the ground literals holds true. If that is not the case, we have found a clash and the DL-safe knowledge base is inconsistent. Thus, for every literal in the rule we create a disjunction branch, in which we assert that the literal holds (in a manner explained in Figure 1 ) - this branch is then explored further. This is the most naive version of the algorithm, and optimizations are discussed in the next sections.

Please note here that for brevity, we do not include all of the $\mathcal{SHOIQ}$ tableaux completion rules (they are defined in [7]). We only present our new expansion rule which is to be applied with lowest priority.

[1]Thanks to the anonymous reviewer for the suggestion

```
$\rightarrow R$ **rule**:
  foreach DNF clause $R_i \in P$:
    foreach ground substitution to $R_i$ that produces a ground clause $G$ :

      foreach monadic ground literal $B(a) \in G$ :
        if $B \notin \mathcal{L}(a)$, then create a new branch where $\mathcal{L}(a) = \mathcal{L}(a) \cup \{B\}$

      foreach dyadic ground literal $B(a, b) \in G$ :
        if $B$ is of type $\neg A(a, b)$ and $\forall A.\neg\{b\} \notin \mathcal{L}(a)$, then
          create a new branch where $\mathcal{L}(a) = \mathcal{L}(a) \cup \{\forall A.\neg\{b\}\}$

        if $B$ is of type $A(a, b)$ and $(a, b) \notin \mathcal{G}$ then
          create a new branch with $\mathcal{G} = \mathcal{G} \cup \{(a, b)\}$ and $\mathcal{L}(a, b) = \{A\}$

        if $B$ is of type $A(a, b)$, $(a, b) \in \mathcal{G}$ and $A \notin \mathcal{L}(a, b)$ then
          create a new branch where $\mathcal{L}(a, b) = \mathcal{L}(a, b) \cup \{A\}$
```

Figure 1: New Expansion Rule

## 4 Implementation and Optimizations

We implemented a proof of concept by extending the open source reasoner Pellet [1]. Obviously, the bottleneck of the algorithm is the non-determinism introduced by creating and exploring tableaux branches for every binding. Thus, as an easy optimization, we check whether, for a particular binding, the rule ground clause is trivially satisfied (one of the disjuncts occurs in the tableaux). If this is the case, we do not generate any new branches for that clause in the tableaux.

More advanced, yet practical, optimizations are possible. For example, we do not need to generate all possible groundings of the rules. Thus, we use a fix-point evaluation procedure as an optimization for our algorithm. By running that procedure first, we make sure that all of the obvious rules are fired - by obvious we mean those rules whose patterns we can match syntactically against the individuals in the completion forest. We use the efficent pattern matching RETE [5] algorithm for this purpose. After the fixpoint is reached, we continue with our tableaux-based algorithm. The speed up comes from the fact that for a given grounding for a rule, if the rule was already fired in the pattern-matching phase, now we do not have to try each disjunct of its horn clause separately, thus decreasing non-determinism.

Another optimization would be to sort the rules after the fix point optimization and before running our algorithm. For a given rule $r$:

$$H \leftarrow B_1, \ldots, B_n$$

we would like to try those groundings that satisfy the maximum amount of conditions $B_i$ first. By doing this we will succeed in firing the 'almost obvious' applications of the rules first, and add more information in the concept labels of the tableaux. In general, the more information we have in the concept labels, the higher the chances to reach a clash when grounding the horn clauses of the rules, yielding less non-determinism in the algorithm.

## 5 Evaluation

We conducted an experimental study to compare the performance of our approach with KAON2 [10]. Our test case of choice was a modified version of the LUBM benchmark [6], with one university and increasing ABox sizes. There are 46 defined classes and 30 object properties in the ontologies. We extended this ontology by adding a rules component consisting of 2 rules:

$GraduateStudent(X)$:-
$Person(X), takesCourse(X,Y), GraduateCourse(Y).$

$SpecialCourse(Z)$:-
$FullProfessor(X), headOf(X,Y), teacherOf(X,Z).$

We ran the experiments on an IBM ThinkPad T42 with Pentium Centrino 1.6GHz processor and 1.5GB of memory. The performance results (time in milliseconds) for the consistency check of the ontologies are shown in table 1. *Rule Branch* stands for the number of additional branches introduced by the $R$-rule.

| No | Size of ontology | Pellet DL-Safe | | KAON2 |
|---|---|---|---|---|
| | | Rule Branches | Consistency Check Time | Consistency Check Time |
| 1 | LUBM, 1 rule, consistent, 17 individuals | 34 | 100 | 661 |
| 2 | LUBM, 1 rule, consistent, 94 individuals | 107 | 250 | 621 |
| 3 | LUBM, 1 rule, inconsistent, 94 individuals | 120 | 381 | 651 |
| 4 | LUBM, 2 rules, consistent, 17 individuals | 109 | 421 | 701 |
| 5 | LUBM, 2 rules, inconsistent, 94 individuals | 422 | 951 | 691 |
| 6 | LUBM, 2 rules, inconsistent, 94 individuals | 0 | 90 | 691 |
| 7 | LUBM, 2 rules, consistent, 94 individuals | 422 | 1232 | 5748 |

Table 1: Evaluation Results for Consistency Checking. Rule Branches stands for the total number of branches introduced by the $R$-rule

Please note that the first rule has 2 variables, and the second has 3. The additional variable in the second rule impacts the performance, since there are considerably more bindings possible for that rule. As an example of the exponential blow up of the bindings, consider that in the case of LUBM with two rules and 94 individuals, in the naive implementation there were 804264 bindings to try. This observation lead us to another optimization strategy - prune the bindings search space. To accomplish this, we embed the trivial satisfiability check in the generation of the bindings. This strategy, along with the implementation of the RETE algorithm as a preprocessor, improved the performance considerably.

We present our results compared to KAON2 in Table 1. As it can be seen, for smaller cases we perform favorably, however the faster rate of increase of consistency check time on our part suggests that for larger ontologies we will be slower than KAON2. The last two test cases merit more discussion. In case of 6), after the RETE algorithm fired the obvious rules we got a clash early and avoided generation of groundings. In the lasr test case, we added another class , `OverachievingProfessor` which is equivalent to a `FullProfessor` who teaches at least 9 courses. This was to demonstrate how we perform better in presence of cardinality constraints.

## 6 Related Work

The most relevant related work to ours is the only other practical reasoning algorithm that processes DL-Safe KBs . The algorithm is described in [11] (implemented in [10]) and is based on the $\mathcal{SHIQ}(D)$ logic and having restricted the DL-atoms in rules to concepts and simple roles. The algorithm is based on reducing the description logic knowledge base to a positive disjunctive datalog program which entails the same set of ground facts as the original knowledge base. As shown in Section 5, our algorithm compares reasonably well for smaller ontologies and for cases when cardinality constraints are used.

## 7 Conclusions and Future Work

This paper makes the following contributions:

- Provides a direct tableaux procedure for the combination of OWL-DL and DL-safe rules.
- Provides preliminary empirical results of an implementation and discussion of possible optimizations.

As for our next steps, we noticed that a the reordering of the literals in the rule makes a noticeable difference in the running time, so we will try to work out the optimum orderings. Also, in longer term, we plan to work on evolving our algorithm toward SWRL by integrating a first order Free Variable tableaux procedure with the $\mathcal{SHOIQ}$ algorithm.

## References

[1] Pellet - OWL DL Reasoner. http://www.mindswap.org/2003/pellet.

[2] Rule interchange format working group, 2005. http://www.w3.org/2005/rules/wg.

[3] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Web Ontology Language (OWL) Reference Version 1.0. W3C Working Draft 12 November 2002 http://www.w3.org/TR/2002/WD-owl-ref-20021112/.

[4] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Al-log: integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10:227–252, 1998.

[5] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[6] J. Heflin, Z. Pan, and Y. Guo. The lehigh university benchmark LUBM. http://swat.cse.lehigh.edu/projects/lubm/, 2003.

[7] I. Horrocks and U. Sattler. A tableaux decision procedure for shoiq. In *Proc. of IJCAI 2005*, pages 448 – 453.

[8] I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic shiq. In D. MacAllester, editor, *Proc. of the CADE 2000*, number 1831, pages 482–496. Springer-Verlag, 2000.

[9] A. Levy and M.-C. Rousset. CARIN: A representation language combining horn rules and description logics. *Artificial Intelligence*, 104(1-2):165–209, 1998.

[10] B. Motik. KAON2 - ontology management for the semantic web. http://kaon2.semanticweb.org/, 2005.

[11] B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univesitt Karlsruhe, Germany, January 2006.

[12] B. Motik, U. Sattler, and R. Studer. Query answering for owl-dl with rules. In *Proc. of ISWC 2004*, pages 549–563.